Data storage: Hardware

Summary so far

- Relational model
- SQL
- Relational Algebra
- Additional features such as triggers, stored procedures, views.

- Basically: the <u>abstraction your RDBMS provides you</u>

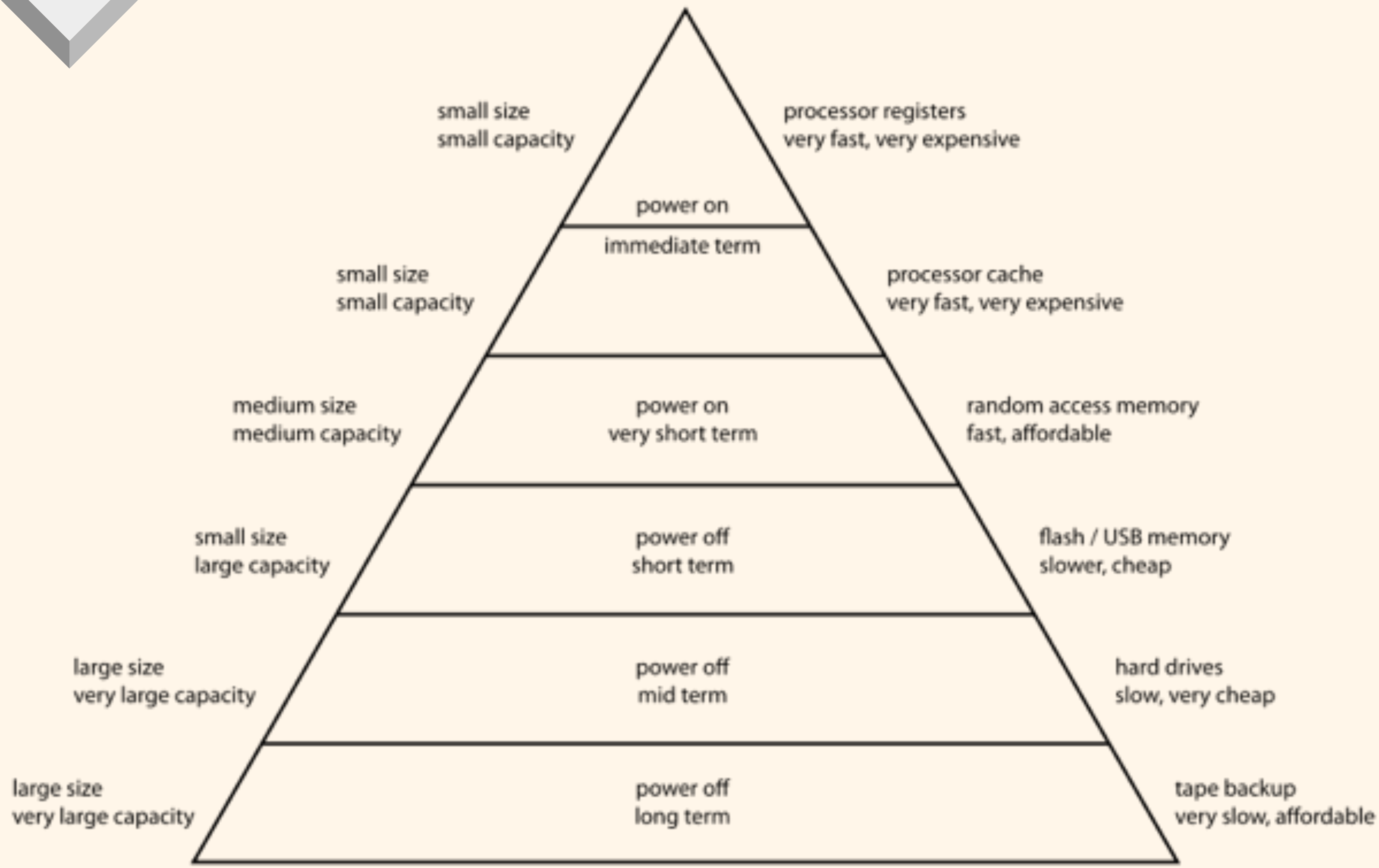
Part 2 (next 3 weeks)

- How is this abstraction actually implemented?
- On the real hardware of your computer…
- Start from the bottom (physical hardware) and get back to relational algebra gradually

Storing and Retrieving Data

- Database Management Systems need to:
  - Store large volumes of data
  - Store data reliably (so that data is not lost!)
  - Retrieve data efficiently
- Alternatives for storage
  - Random Access Memory
    - DRAM, SRAM, (soon:) PCM, Memristor
  - Disks
    - Traditionally HDDs, now SSD and hybrid solutions
  - Tape, optical media, …

Computer Memory Hierarchy



Some numbers

Level	Access time	Typical size
Registers	"instantaneous"	under 1KB
Level 1 Cache	1-3 ns	64KB per core
Level 2 Cache	3-10 ns	256KB per core
Level 3 Cache	10-20 ns	2-20 MB per chip
Main Memory	30-60 ns	16-64 GB per system
Hard Disk	3,000,000-10,000,000 ns	over 1TB

Problem

- CPU sits idle most of the time waiting for your HDD to deliver data!
- Will SSDs solve that problem?
- They are helping, but
  - Still a relatively new and evolving technology
  - Flash still slower than DRAM
  - Can't beat hard disks for storage capacity and price

Solutions (1)

- Physically make storage faster
  - still happening with HDDs,
  - or use new tech like SSDs
  - or even have a main-memory DB (!!!)
- Redundant Array of Inexpensive Disks (RAID)
  - Achieve parallelism by using many disks
- Intelligent data layout on disk
  - Put related data items together

Solutions (2)

- As with everything else in computing, *cache.*
- Keep “currently used” data in main memory
  - How do we do this efficiently?
  - OS provides virtual memory support
    - But this may not be the best solution for the custom access patterns that a DBMS has

Remainder of today's lecture

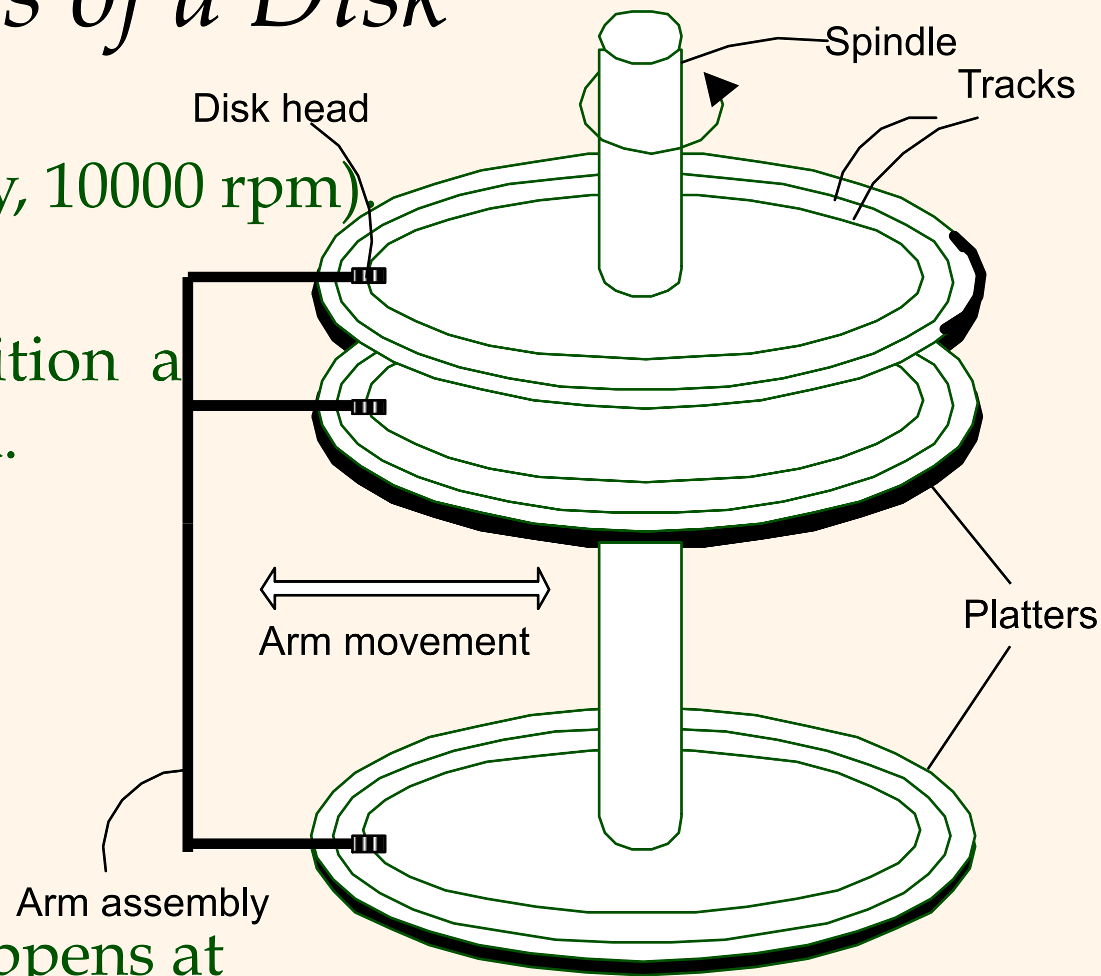
- Brief discussion of the hardware your data is stored on
- Not comprehensive, not in-depth
- Just enough to familiarize you with
  - The hardware abstraction we need
  - The performance model(s) associated with this hardware abstraction
    - What operations are cheap/expensive

Hard disks

- Tech has been around since 1956, with the IBM 305 RAMAC
- http://www.youtube.com/watch?v=zOD1umMX2s8

Components of a Disk

- The platters spin (say, 10000 rpm).
- The arm assembly is moved in or out to position a head on a desired track.



- Reading/writing happens at the granularity of a *block*

Logical Block Addressing

- Geometry of disk at the cylinder/head level abstraction typically no longer exposed to software
- Instead, device provides abstraction of a set of blocks which are indexed and accessed linearly (block 0, block 1, etc.)
- Standard block ("sector") size : 4K

Accessing a Disk Block

- Time to access (read/write) a disk block:
  - *seek time* (moving arms to position disk head on track)
  - *rotational delay* (waiting for block to rotate under head)
  - *transfer time* (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 10msec
  - Rotational delay varies from 0 to 5msec
  - Example transfer rate: 600MB/s max (135 MB/s sustained) (from a current Seagate laptop's drive specs)
- Key to lower I/O cost: reduce seek/rotation delays!

Arranging Data on Disk

- Blocks in a file should be arranged sequentially on disk, to minimize seek and rotational delay.
  - If you've ever run defrag and it improved performance, this is why
- For a sequential scan, *pre-fetching* several blocks at a time is a big win!

RAID

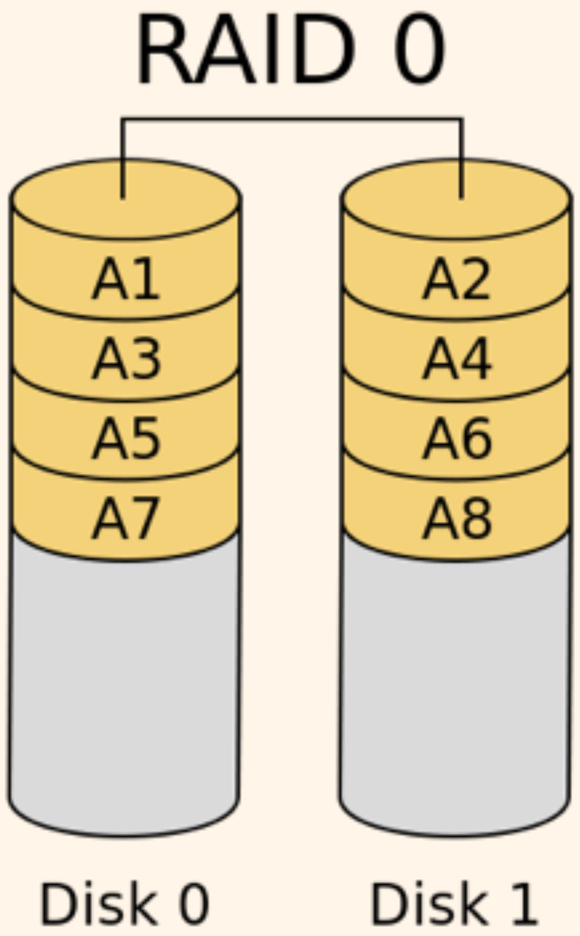
- Redundant Array of Inexpensive Disks
- Arrangement of several disks that gives abstraction of a single, large disk.
- Goals: Increase performance and reliability.
- Two main techniques:
  - Data striping: Data is partitioned; size of a partition is called the striping unit. Partitions are distributed over several disks.
  - Redundancy: More disks -> more failures. Redundant information allows reconstruction of data if a disk fails. Two main approaches: parity and mirroring.

Parity

- Add 1 redundant block for every n blocks of data
  - XOR of the n blocks
- Example: D1, D2, D3, D4 are data blocks
  - Compute DP as D1 XOR D2 XOR D3 XOR D4
  - Store D1, D2, D3, D4, DP on different disks
  - Can recover any *one* of them from the other four

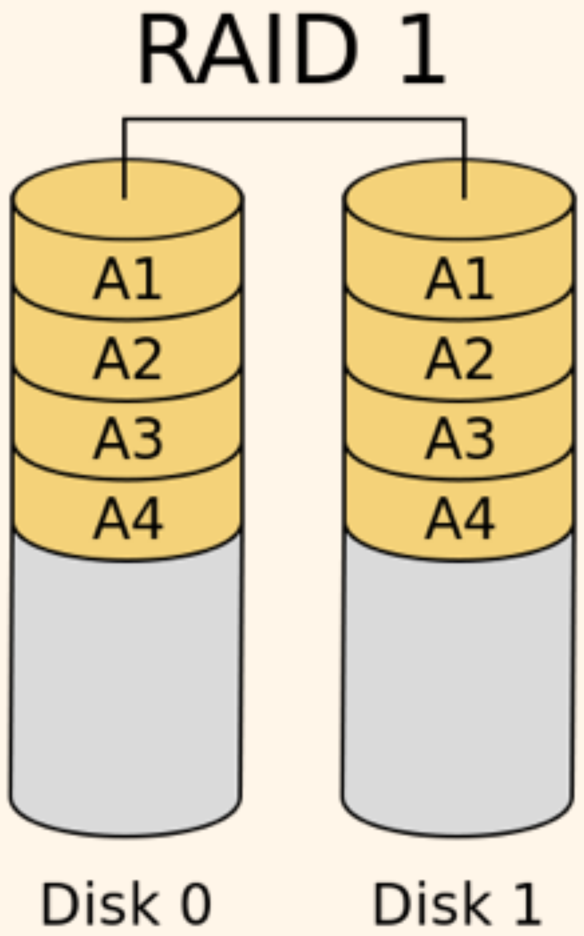
RAID Level 0

- No redundancy
  - Striping without parity
  - Striping at block level



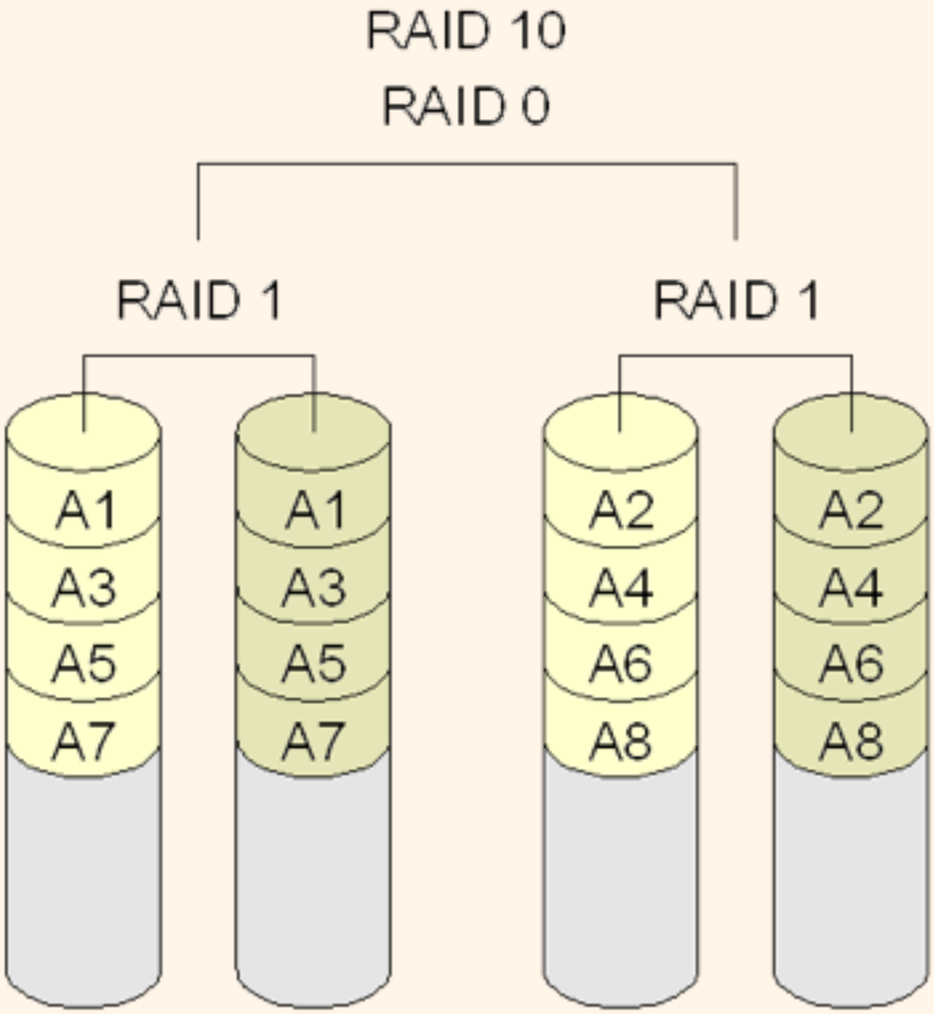
RAID Level 1

- Level 1: Mirrored (two identical copies)
  - Each disk has a mirror image (check disk)
  - Parallel access for reads
  - Write involves two disks.



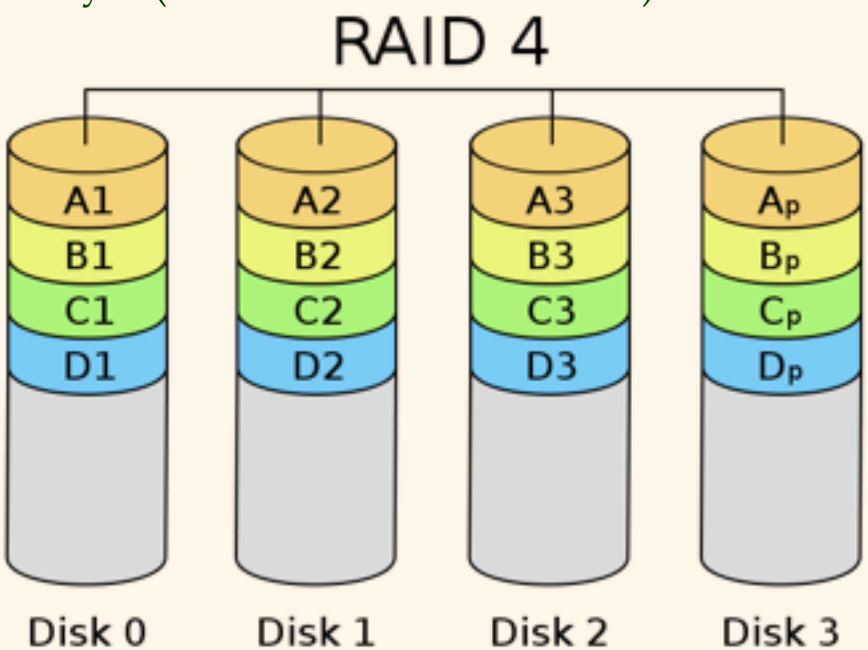
RAID Level 0+1 (or 10)

- Striping and Mirroring
  - Parallel reads.
  - Write involves two disks.
  - Combines performance of RAID 0 with redundancy of RAID 1.



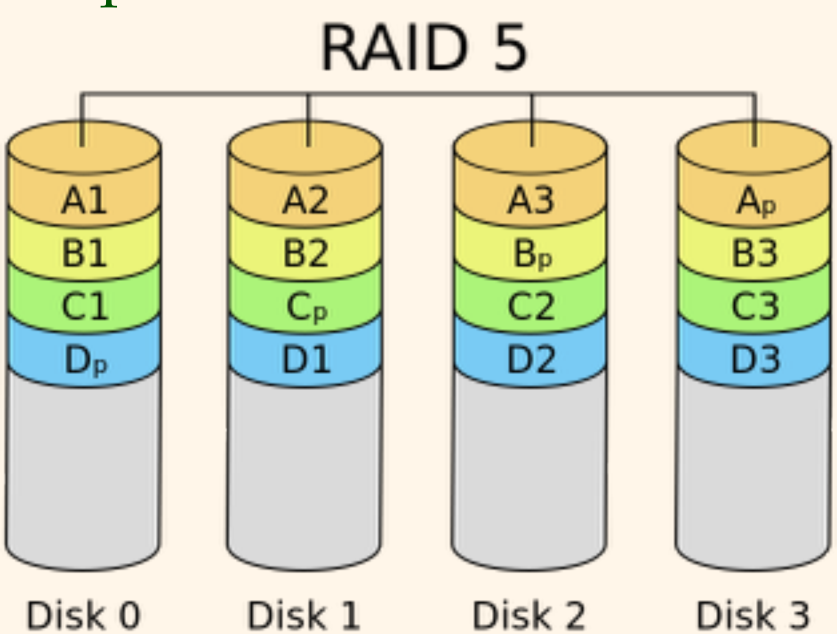
RAID Level 4

- Block-Interleaved Parity
  - Striping Unit: One disk block. One check disk.
  - Parallel reads possible for small requests, large requests can utilize full bandwidth
  - Writes involve modified block and check disk
    - New Parity = (Old Data XOR New Data) XOR Old Parity



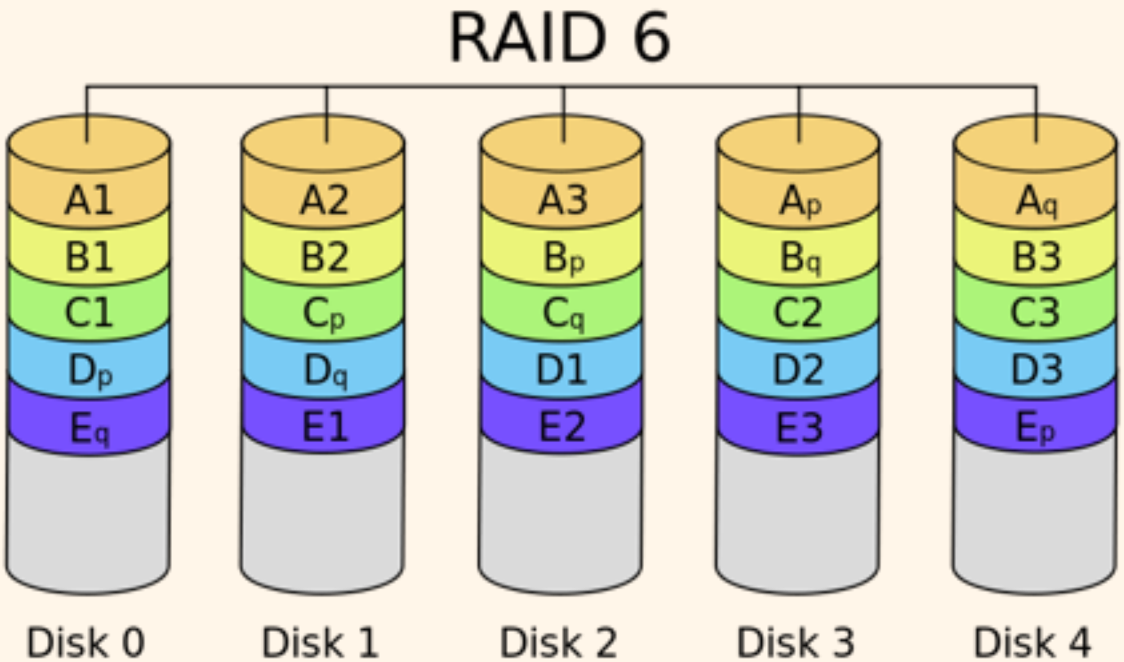
RAID Level 5

- Block-Interleaved Distributed Parity
  - Similar to RAID Level 4, but parity blocks are distributed over all disks
  - Eliminates check disk bottleneck, one more disk for higher read parallelism



RAID Level 6

- Use a second function in addition to parity
  - Can recover from 2 failed disks



SSDs

- A storage medium gaining in popularity
- Made from NAND flash memory (like your USB flash drive)
- But much faster than your USB flash drive due to clever engineering in the SSD's controller
- Fast evolving proprietary technology
  - http://arstechnica.com/information-technology/2012/06/inside-the-ssd-revolution-how-solid-state-disks-really-work/

NAND flash memory

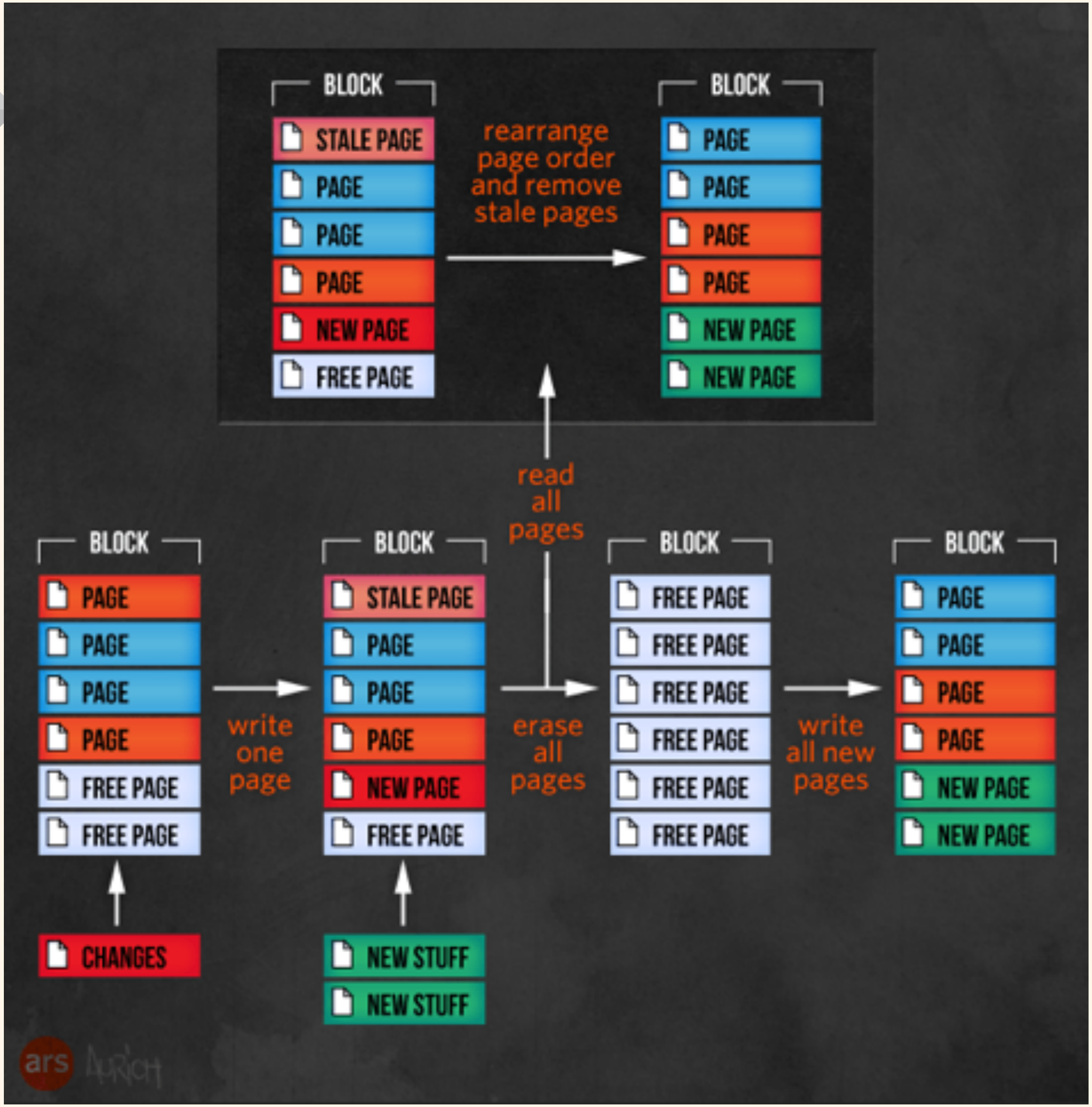
- Technical details beyond the scope of this class
- But should know that absolute min granularity of access is a *page*
  - Typically, 4K or 8K
- A *block* consists of a number of *pages*
  - Say 32, 64, 128 or even 256 pages
  - So a block could be 1M or more

NAND flash memory

- Can read or write to individual pages, but
- Cannot *overwrite* pages that already contain data
  - If a page is blank and contains all 1's, easy to turn certain 1's into 0's
  - But turning 0's into 1's tricky and hard to confine to the bits of interest

Block level erasure

- To write, need to find a freshly erased page
- If none available, need to erase *an entire block*
- Then write the erased block with its old contents and the new page


BLOCK
STALE PAGE
PAGE
PAGE
PAGE
NEW PAGE
FREE PAGE
rearrange page order and remove stale pages
BLOCK
PAGE
PAGE
PAGE
PAGE
NEW PAGE
NEW PAGE
read all pages
BLOCK
PAGE
PAGE
PAGE
PAGE
FREE PAGE
FREE PAGE
CHANGES
write one page
BLOCK
STALE PAGE
PAGE
PAGE
PAGE
NEW PAGE
FREE PAGE
NEW STUFF
NEW STUFF
erase all pages
BLOCK
FREE PAGE
FREE PAGE
FREE PAGE
FREE PAGE
FREE PAGE
FREE PAGE
write all new pages
BLOCK
PAGE
PAGE
PAGE
PAGE
NEW PAGE
NEW PAGE
ars

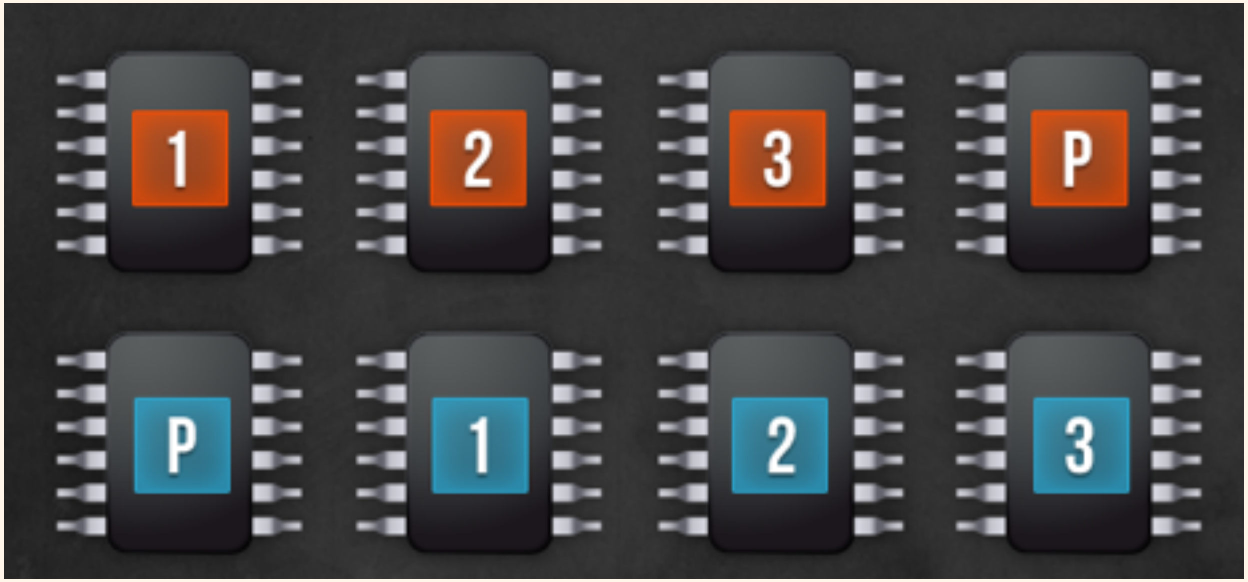
Longevity issues

- NAND flash can only take a limited number of write cycles before it degrades
  - Somewhere between 1K and 100K cycles
- Not an issue for your laptop (will probably replace the laptop before you hit that limit)
- But in the enterprise/datacenter, workloads on disks very high
- Lots of clever engineering in SSDs to work around that

Making SSDs fast and long-lived

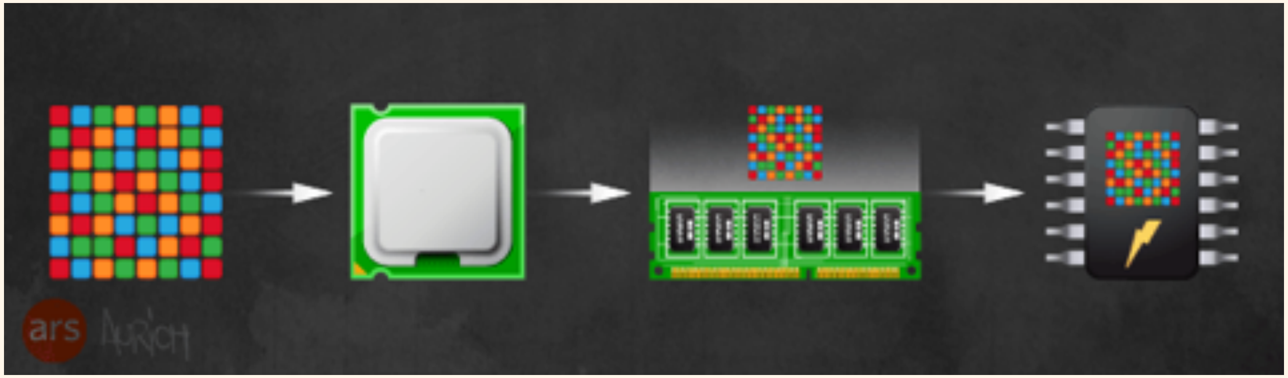
- Typical techniques:
  - Striping for performance (cf. RAID)
  - Error correction (cf. RAID)
  - Use a DRAM cache in front of the SSD
  - Over-provisioning
  - Garbage collection
  - TRIM command
  - Wear leveling

"RAID 5" in your SSD



- Usually called something slightly more buzzwordy, like RAIN or RAISE

Caching in an SSD



- Cache holds data to be written until SSD can take a write
    - Obviously need to worry about power loss
    - But in the enterprise, that may not be such a likely event

Over-provisioning

- Your SSD actually contains more capacity than "advertised"
- Extra space handy to maximize chance there will be a free (erased) block to write to
- And when some cells start to fail due to too many write cycles, will still have more good ones

Garbage collection

- Keep track of stale pages
- Periodically erase a block and compact non-stale data into that block
- Obviously this takes up one of the limited number of erase cycles though, so the most aggressive GC strategy not always optimal

TRIM

- Normally, when you "delete" a file, the data is not removed from your hard disk, the OS just updates its metadata in file system
  - But the disk doesn't know anything about this (or even what OS is in use)
- With TRIM command, OS tells SSD page is stale
  - And does not need to be retained during garbage collection

Wear leveling

- SSD controller tries to ensure each cell gets a similar number of writes
- So, periodically swaps around "hot data" that gets written a lot with "cold data"
- But this of course takes write/erase cycles away from the lifetime of the drive…

Write amplification

- Due to the *block erase* requirement and the various techniques we discussed, writing x bytes of data may require writing much more than x bytes to the SSD
- Which is bad for performance and longevity
- Reducing write amplification an ongoing subject of research

SSDs vs HDDs

- Random reads are much faster
  - Don't need to worry about data being contiguous for fastest reads
  - No need to defrag
- Writes – not always equally fast, due to block erase and write amplification more generally
- Also hybrid SSD/HDD solutions out there ("SSHDs")

SSDs for databases

- Best way to use SSDs for data storage depends on the specific functionality of the controller
- Very much a moving target
- A lot of interest (and startups!)

In this course

- We will discuss how to implement relational algebra operators using a simple **performance model** for HDDs
- Some of this is still applicable to SSDs
- Whatever storage device you will be using in 10/20/50 years' time, it will still have limitations, pros and cons that you need to take into account
  - You can see our use of HDDs as a case study